IFW

PTO/SB/21 (08-00)
Approved for use through 10/31/02. OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMENRCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays avalid OMB control number.

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	10/626,419
Filing Date	July 23, 2003
First Named Inventor	Pierre Lebee, et al.
Group Art Unit	2183
Examiner Name	NYA
Total Number of Pages in This Submission: 22	Attorney Docket Number: 15437-0621

ENCLOSURES ***(check all that apply)***

- ☐ Fee Transmittal Form
  - ☐ Fee Attached
- ☐ Amendment / Response
  - ☐ After Final
  - ☐ Affidavits/declaration(s)
- ☐ Extension of Time Request
- ☐ Express Abandonment Request
- ☐ Information Disclosure Statement
- ☒ Certified Copy of Priority Document(s)
  from France dtd 24 Avr. 2003
- ☐ Response to Missing Parts/ Incomplete Application
  - ☐ Response to Missing Parts under 37 CFR 1.52 or 1.53
- ☐ Assignment Papers *(for an Application)*
- ☐ Drawing(s)
- ☐ Licensing-related Papers
- ☐ Petition
- ☐ Petition To Convert To a Provisional Application
- ☐ Power of Attorney, Revocation Change of Correspondence Address
- ☐ Terminal Disclaimer
- ☐ Request for Refund
- ☐ CD, number of CD(s)_____
- ☐ After Allowance Communication to Group
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- ☐ Proprietary Information
- ☐ Status Letter
- ☐ Other Enclosure(s) *(please identify below):*

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

Firm or Individual name	Hickman Palermo Truong & Becker LLP Bobby K. Truong
Signature	
Date	June 3, 2004

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class: mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: 6/3/04

Type or printed name	Annette Jacobs		
Signature		Date	June 3, 2004

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

R E P U B L I Q U E F R A N Ç A I S E

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 24 AVR. 2003

Pour le Directeur général de l'Institut
national de la propriété industrielle
Le Chef du Département des brevets

Martine PLANCHE

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
SIEGE
26 bis, rue de Saint Petersbourg
75800 PARIS cedex 08
Téléphone : 33 (0)1 53 04 53 04
Télécopie : 33 (0)1 53 04 45 23
www.inpi.fr

ETABLISSEMENT PUBLIC NATIONAL CREE PAR LA LOI N° 51-444 DU 19 AVRIL 1951

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 01 53 04 53 04 Télécopie : 01 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ
Code de la propriété intellectuelle - Livre VI

cerfa N° 11354*01

REQUÊTE EN DÉLIVRANCE 1/2

Important ! Remplir impérativement la 2ème page.

Cet imprimé est à remplir lisiblement à l'encre noire DB 540 W /190600

Réservé à l'INPI

REMISE DES PIÈCES DATE 23 JUIL. 2002
LIEU 75 INPI PARIS
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0209343
DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI 23 JUIL. 2002

Vos références pour ce dossier *(facultatif)* SUN Aff. 26 (120633)

1 NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE

CABINET NETTER
36 avenue Hoche
75008 PARIS

Confirmation d'un dépôt par télécopie ☐ N° attribué par l'INPI à la télécopie

2 NATURE DE LA DEMANDE	Cochez l'une des 4 cases suivantes
Demande de brevet	☒
Demande de certificat d'utilité	☐
Demande divisionnaire	☐
Demande de brevet initiale	N° Date / /
ou demande de certificat d'utilité initiale	N° Date / /
Transformation d'une demande de brevet européen *Demande de brevet initiale*	☐ N° Date / /

3 TITRE DE L'INVENTION (200 caractères ou espaces maximum)

Non intrusive testing method for real-time software.

4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE	
	Pays ou organisation Date / / N°
	Pays ou organisation Date / / N°
	Pays ou organisation Date / / N°
	☐ **S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»**

5 DEMANDEUR		☐ **S'il y a d'autres demandeurs, cochez la case et utilisez l'imprimé «Suite»**
Nom ou dénomination sociale		SUN MICROSYSTEMS, INC
Prénoms		
Forme juridique		
N° SIREN		
Code APE-NAF		
Adresse	Rue	901 San Antonio Road
	Code postal et ville	94303 PALO ALTO Californie
Pays		Etats-Unis d'Amérique
Nationalité		Société des Etats-Unis d'Amérique
N° de téléphone *(facultatif)*		
N° de télécopie *(facultatif)*		
Adresse électronique *(facultatif)*		

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

REQUÊTE EN DÉLIVRANCE 2/2

Réservé à l'INPI

REMISE DES PIÈCES
DATE 25 JUIL 2002
LIEU 75 INPI PARIS
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0209343

DB 540 W /190600

Vos références pour ce dossier : *(facultatif)*		SUN Aff. 26 (120633)
6 MANDATAIRE		
Nom		PLACAIS
Prénom		Jean-Yves
Cabinet ou Société		Cabinet NETTER
N °de pouvoir permanent et/ou de lien contractuel		
Adresse	Rue	36 avenue Hoche
	Code postal et ville	75008 PARIS
N° de téléphone *(facultatif)*		01 58 36 44 22
N° de télécopie *(facultatif)*		01 42 25 00 45
Adresse électronique *(facultatif)*		
7 INVENTEUR (S)		
Les inventeurs sont les demandeurs		☐ Oui ☒ Non **Dans ce cas fournir une désignation d'inventeur(s) séparée**
8 RAPPORT DE RECHERCHE		**Uniquement pour une demande de brevet (y compris division et transformation)**
Établissement immédiat ou établissement différé		☐ ☒
Paiement échelonné de la redevance		**Paiement en deux versements, uniquement pour les personnes physiques** ☐ Oui ☐ Non
9 RÉDUCTION DU TAUX DES REDEVANCES		**Uniquement pour les personnes physiques** ☐ Requise pour la première fois pour cette invention *(joindre un avis de non-imposition)* ☐ Requise antérieurement à ce dépôt *(joindre une copie de la décision d'admission pour cette invention ou indiquer sa référence)* :
Si vous avez utilisé l'imprimé «Suite», indiquez le nombre de pages jointes		

10 SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire)

Paris, le 22 juillet 2002
N° Cnseil 92-1197 (B) (M)

VISA DE LA PRÉFECTURE OU DE L'INPI

L. MARIELLO

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire. Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

July 19, 2002 10:00:28 Sun26.FRD.wpd

P6543

Non intrusive testing method for real-time software

This invention generally relates to the computer technology, more precisely to debugging methods.

Robust operating systems require to be debugged, e.g. to detect problems in execution on operating systems or to improve configuration of the operating systems. Moreover, applications require also to be tested. More generally, all software applications and software tools require debugging methods.

It exists some testing methods which do not keep traces of non trivial problems during and/or after execution, e.g. debugging methods which display traces over a console, debugging methods which proposes breakpoints. These methods do not enable a user to analyze a posteriori the traces of non trivial problems.

To keep traces of non trivial problems, debugging method may require an important amount of memory. Moreover, testing real-time and asynchronous software is particularly complex because the debugging execution is generally not deterministic, in other words time dependant. These aspects modify the behavior of the debugged operating system, applications or software tools. The debugging execution is thus very intrusive.

An external hardware technology, as ICE (In Circuit Emulator) emulator technology, is known to be adapted to retrieve data on a bus, e.g. between a CPU and a memory, such data corresponding to the execution of operations. This hardware technology is thus non intrusive. However, such data may not be retrieved by the emulator in the order of the execution of the operations. Moreover, some data corresponding to the execution of operations are not communicated through the bus but stay in the CPU, e.g. in register of the CPU. Thus, this hardware technology has a major problem as it does not enable a user to access to all data corresponding to the execution of operations and to analyze the retrieved data. Detection of debugging problems is thus impossible.

A general aim of the present invention is to provide advances with respect to such mechanisms.

The invention concerns a debugging framework intended to receive data of a debug operation, said debugging framework comprising

- a reserved memory divided into a selected number of portions of reserved memory,
- a mass memory,
- a log management component adapted to execute a first function capable of recording received data of a debug operation in portions of a first memory available to store data and, responsive to drain conditions, a second function capable of copying said data from at least a partially filled portion of reserved memory to the mass memory, so as to keep successive received data of a debug operation in a non intrusive way.

The invention also concerns a method of managing data of a debug operation, comprising the steps of:

a. reserving a memory divided into a selected number of portions of memory,

b. recording received data of a debug operation in portions of memory available to store data and

c. responsive to drain conditions, copying said data from at least partially filled portions of memory to a mass memory

d. repeat steps b. and c. while data of a debug operation are received so as to keep successive received data of a debug operation in a non intrusive way.

Thus the invention permits to record in a non intrusive way the effective behavior of, e.g. an application, for further off-line analysis.

Other alternative features and advantages of the invention will appear in the detailed description below and in the appended drawings, in which :

- figure 1 is a general diagram of a computer system in which the invention is applicable;

- figure 2 shows a partial diagram of the computer system according to the invention;

- figure 3 is a general view of a portion of a working memory according to the invention;

- figure 4 is a view of the portion of the working memory of figure 3 in a operation of a record method;

- figure 5 is another view of the working memory of figure 4 in an other operation of the record method;

- figure 6 is another view of the working memory of figure 5 in an other operation of the record method;

- figure 7 is a flow chart of the record method;

- figure 8 is a flow chart of the record method according to the invention;

- figure 9 is a flow chart of the drain method according to the invention.

As they may be cited in this specification, Sun, Sun Microsystems, Solaris, ChorusOS are trademarks of Sun Microsystems, Inc. SPARC is a trademark of SPARC International, Inc.

In figure 1, this invention may be implemented in a computer system, or in a network comprising computer systems. The hardware of such a computer system is for example as shown in Fig.1, where:

- 11 is a processor, e.g. an Ultra-Sparc;
- 12 is a program memory, e.g. an EPROM for BIOS, a RAM, or Flash memory, or any other suitable type of memory;
- 13 is a working memory, e.g. a RAM of any suitable technology (SDRAM for example);
- 14 is a mass memory, e.g. one or more hard disks;
- 15 is a display, e.g. a monitor;

- 16 is a user input device, e.g. a keyboard and/or mouse; and
- 21 is a network interface device connected to a communication medium 20, itself in communication with other computers. Network interface device 21 may be an Ethernet device, a serial line device, or an ATM device, inter alia. Medium 20 may be based on wire cables, fiber optics, or radio-communications, for example.

Data may be exchanged between the components of figure 1 through a bus system 10, schematically shown as a single bus for simplification of the drawing. As is known, bus systems may often include a processor bus, e.g. of the PCI type, connected via appropriate bridges to e.g. an ISA bus and/or an SCSI bus.

In the following description, to reserve memory means to pre-allocate memory for a particular purpose.

The foregoing description relates to figures 2 and 3. Figure 2 represents a log system 50 comprising a portion of working memory 131, a log management component 600 and a non volatile storage unit 14. The portion of working memory 131, also called a log and comprised in the RAM 13, is divided in N portions of memory, as chunk 1 to chunk N also designated as C_1 to C_N. A log is a fixed size record containing unsigned integers. Each chunk is advantageously divided in M sub-portions of memory called log items, e.g. C_8 is divided in M log items from L_1-C_8 to L_M-C_8 and C_j comprises log item i designated as L_i-C_j in figure 3. A log item may also be a portion of memory. Each log item may comprise
- a log stamp, being an unsigned integer, to designate a number incremented at each record of the log item,
- a serie of p log values, which may be unsigned integers, p being an integer. This integer p may be fixed by a user and may provide a fixed size for the log item.

At least one log is reserved before the start of a debugging method. This log comprises several chunks, each chunk being divided into several log items. In each log item, the log management component is adapted to store debugging data, these debugging data are stored in the serie of log value. The log management component 600 comprises a record function

α

610 to record debugging data in log items and a drain function 620 to copy debugging data from log items to the non volatile storage unit 14, also called a mass memory. As described hereinafter, the drain function 620 may be started or its execution may continue if drain conditions 630 are satisfied. To start the drain function, these drain conditions may comprise a request of the user to use the drain function. These drain conditions may define a variable that indicates the drain function is required or not. As seen hereinafter, these drain conditions may also comprise other conditions.

In the following description, data are considered to be debugging data, e.g. data concerning information which may be considered as useful to debug a program, e.g. program of an operating system. A programmer may choose to retrieve debugging data by inserting instructions.

A log item may be in different states :

- "Available to store data" if no data has been stored (or recorded) in the log item or if data stored in the log item have been copied (or transferred) from the log item, also called a data transfer in the following description,
- "filled of data" if data have been stored in the log item.

A chunk may also be in different states :

- "Available to store data" if no data has been stored (or recorded) in the chunk or if data stored in the chunk have been copied (or transferred) from the chunk, also called a data transfer in the following description,
- "filled of data" if data have been stored in the chunk. For a chunk, "filled of data" comprises "completely filled of data" if all log items of the chunk are filled of data and "partially filled of data" if some log items of the chunk are filled of data.

A log may also be "completely filled of data" or "partially filled of data".

In figure 2, cross-hatched in solid lines chunks as C_1, C_{k+1}, and C_N designate chunks having log items filled of data. C_j is composed of different log items: cross-hatched log items are

filled of data, the cross-hatched in dotted lines log item L_i-C_j represents a log item in a data record phase, other non cross-hatched log items represent log items available to store data and thus available for a data record phase. Other non cross-hatched chunks as C_{j+1} to C_k represent chunks available to store data.

The log is linked to the log management component 600 adapted to manage a record method in the log. Two methods of data management are hereinabove described, a variable called a flag designating the chosen method, e.g. chosen by a user. First method is designated as "record only method", second method is designated as "record and drain method". In second method, this log management component 600 is adapted to manage a data transfer method from the log to the non volatile storage unit 14, e.g. a hard disk.

Figure 3 illustrates a log at the initialization time before recording data in the log. In other words, all log items are available to store data.

Figure 4 illustrates an instant of data recording according to the " record only method" or the "record and drain method" of the invention.

Thus, in figure 4, the log comprises N chunks C_1 to C_N. At the beginning of an application, all the chunks are available to store data. The log management component is adapted to store data in the log according to the operations of a record method. The log management component stores first data in the log items of the chunk C_1. The arrow B designates the log item in a data record phase. In figure 4, this arrow B corresponds to the log item L_1-C_1 in a data record phase. In the record only method, the entire log is filled with data, arrow B designates the limit between a first part of the log being filled of data and a second part of the log being available to store data. In the record and drain method, this arrow B designates the front end of the portion of log containing log items filled of data, a second arrow designating the back end of this portion of log as illustrating in figure 5. A portion of log designates a given number of chunks.

In the record only method, the log management component stores data in the successive log items of the chunks C_1 to C_N. Once the log is completely filled of data, the record only method enables to reuse a log item of a chunk in order to record next arriving data. The next log item to be reused, and thus erased, is the log item having the oldest recorded data. As in this example data are recorded in consecutive log items, the oldest recorded data is in L_1-C_1 when the log is completely filled and is a circular buffer. Then, the oldest recorded data is in the consecutive log items or in the first log item as the log is a circular buffer.

Figures 5 to 6 illustrate different instants of data record method according to the record and drain method of the invention.

In figure 5, the log management component stores data in the successive log items of the chunks C_1 to C_3. In other words, when the first chunk is completely filled of data, the log management component stores data in the second chunk. The arrow B corresponds to the log item L_j-C_3 in a data record phase. The log management component is also adapted to transfer data of log items in the non volatile storage unit. An arrow A designates the limit between a log item whose data has just been transferred in the non volatile storage unit and log items filled of data. Thus, log items filled of data are limited at a first extremity by the arrow A and at the other extremity by a second arrow B.

In figure 6, the log management component continues to store data in the successive log items of the chunks C_3 to C_5. The arrow B designates the limit between the first log item of the chunk C_6 and the second log item of the chunk C_6, the arrow A designates the limit between the last log item of the chunk C_2 and the first log item of the chunk C_3. Thus, the log management component has copied successively data from the log items of the first two chunks C_1 to C_2 to the non volatile storage unit while recorded successively other input data in chunks C_3 to C_5.

In the record and drain method, the transfer (drain) method is also called the log consumer method, the record method is also called the log producer method.

During the record method execution, it is checked if the following drain conditions are true to launch the drain method. These drain conditions comprises filling threshold. Thus, the drain conditions may comprise reaching a determined number of filled log items Nli, or reaching a determined number of completely filled chunks Nc. For example, in the implementation of the invention, the drain conditions are true if one chunk is filled of data, meaning if M successive log items are filled of data Nc=1, Nli=M.

According to the application execution profile, Nc, Nli, N and M integers are chosen at the initialization so as to enable the speed of the drain method to be smaller than the speed of the record method. These chosen integers also enable the speed of the drain method to be high enough so as to enable the record method to store data on log items available. Thus, no data is lost. p may be set by default e.g. p may be smaller than 10. p may be also dynamically chosen to enable a record of the more important data in a log item in the least intrusive way.

If these drain conditions are false, the record and drain method stops.

The log system is a generic mechanism adapted to record data in log items in a non-intrusive and deterministic way. As seen, the log with its chunks having log items used to store data is reserved, so that real-time constraints are satisfied. As described, a log is a fixed size record containing unsigned integers. This minimizes the CPU time and the amount of memory needed to store a data in a log item.

The flexible consumer-producer mechanism permits to minimize disruption when data is copied from chunks.

The algorithm has a deterministic behavior allowing to be also invoked at interrupt level.

The record only method is illustrated in figure 7. The record and drain method is illustrated in figure 8 and 9. Figure 8 illustrates the record method of the record and drain method of the invention and figure 9 illustrates the drain (transfer) method of the invention.

In figure 7, a log may only be divided into log items. A data is received to be stored in the log. The record only method starts at operation 100. At operation 110, a pointer designates a new log item available to store the received data. Data is stored in this log item being a filled log item at operation 130.

If no more log item available to store data is left in the log at operation 135, the pointer designates the log item having the oldest recorded data. As the log is circular, the first log item of the log has the oldest recorded data and the pointer designates this first log item to store data at operation 150.

If at least one log item available to store data is left in the log at operation 135, the pointer designates the next log item available to store data at operation 160.

After operations 150 or 160, the method to record the log item ends at step 180 until another data to record is received and the method restarts at step 100.

Figure 8 illustrates the record method of the record and drain method that a user may choose. In figure 8, a log may be divided into chunks and log items. A data is received to be stored in the log. The record method starts at operation 300. At operation 305, it is checked if the log is already full or not. If it is, the method ends at operation 390. Else, the method continues at operation 310.

At operation 310, a pointer designates a new log item available to store the received data. Data is stored in this log item being a filled log item at operation 330.

If at least one log item remains available to store data in the chunk at operation 335, the pointer designates the next log item available to store data of said chunk at operation 360.

If no more log item available to store data is left in the log at operation 335, the drain method of figure 9 begins at operation 340 to drain the completely filled chunk. In another embodiment, as seen, the drain method may be launched when more than one filled chunk

is completely filled of data. At the same time, it is checked at operation 350 if the entire log is completely filled of data.

If the log is entirely filled of data, the record and drain method ends at operation 390.

If at least one chunk is left, this new chunk is get at operation 352. The pointer designates the next log item available to store data in said new chunk at operation 360.

At operation 360, the record method of the record and drain method ends after operation 380. Ending with operation 380, if other data are to be stored in the log, the record method restarts at operation 300.

A flag may be a variable which designates the record only method or the record and drain method. Thus, the record and drain method may be designated with a flag having a "drain" value.

If the record and drain method is designated, the operation 140 is executed to drain detected completely filled chunk(s) as illustrated in figure 8. Thus, the operation 152 gets a chunk available to store data, as this chunk has previously been drained by the transfer (drain) method at operation 140 as described hereinabove.

In figure 8, the flow chart illustrates a method to drain chunks. The flow-chart is waiting for a chunk to be drained at operation 210. When at least one chunk has to be drained at operation 200, the flow chart is activated. In an embodiment, detected completely filled chunks are given as parameters from the record method. At operation 220, a first completely filled chunk is released in the log. According to parameters, other completely filled chunks may be released at operation 230. In this case, an iteration of operations 220 and 230 may be performed for these other completely filled chunks. Else, the method returns to operation 210 in a waiting state.

This drain method is performed in parallel to the record method in a permanent manner.

α

This hereinbefore described method is not intrusive meaning it does not modify the behavior of the system being debugged or application. Indeed, the corresponding framework enables a copy (or transfer) of data from a log item to the mass memory during a debugging method enabling a non intrusive method.

The invention is not limited to the hereinabove described features.

Thus, other embodiments of such record methods or release methods may be developed according to the invention. Moreover, the method of the invention is not limited to the application of managing debugging data using a log management component. Any other type of data may be recorded and managed according to the method of the invention. Thus, this method may be useful in any computer system requiring a non-intrusive method and framework to store a whole historic of data, e.g. in avionic software.

Claims

1. A debugging framework intended to receive data of a debug operation, said debugging framework comprising

- a reserved memory (131) divided into a selected number (N) of portions (Cj) of reserved memory (C_j),
- a mass memory (14),
- a log management component (600) adapted to execute a first function (610) capable of recording received data of a debug operation in portions of a first memory available to store data and, responsive to drain conditions, a second function (620) capable of copying said data from at least a partially filled portion of reserved memory to the mass memory (14), so as to keep successive received data of a debug operation in a non intrusive way.

2. The debugging framework of claim 1, wherein the drain conditions (630) comprise reaching a filling threshold.

3. The debugging framework of claim 2, wherein reaching a filling threshold comprises reaching a determined number of portions of reserved memory which are completely filled of data.

4. The debugging framework of claim 3, the determined number of portion of reserved memory equal to one.

5. The debugging framework of claim 1, wherein the log management component (600) is further adapted to execute the second function responsive to a request of the user.

6. The debugging framework of claim 1, wherein the log management component (600) is adapted to execute the first (610) and second (620) functions in parallel.

7. The debugging framework of claim 1, wherein the portion of memory (Cj) comprises a selected number (M) of sub-portions of memory (Li-Cj).

8. The debugging framework of claim 7, wherein a sub-portion of memory (Li-Cj) comprises a determined number (p) of data value to store at a given time.

9. The debugging framework of claim 8, wherein the determined number (p) of data value is chosen by the user so as to enable a record of the more important data in a log item in the least intrusive way.

10. The debugging framework of claim 9, wherein the determined number (p) of data value is smaller than 10.

11. A method of managing data of a debug operation, comprising the steps of:
a. reserving a memory divided into a selected number of portions of memory,
b. recording received data of a debug operation in portions of memory available to store data and
c. responsive to drain conditions, copying said data from at least partially filled portions of memory to a mass memory
d. repeat steps b. and c. while data of a debug operation are received so as to keep successive received data of a debug operation in a non intrusive way.

12. The method of claim 11, wherein the drain conditions of step c. comprise reaching a filling threshold.

13. The method of claim 12, wherein reaching a filling threshold comprises reaching a determined number of portions of reserved memory which are completely filled of data.

14. The method of claim 13, wherein the determined number of portion of reserved memory equal to one.

15. The method of claim 11, wherein step c. is further adapted to be execute responsive to a request of the user.

16. The method of claim 11, wherein steps b. and c. are executed in parallel.

17. The method of claim 11, wherein the portion of memory (Cj) of step a. comprises a selected number (M) of sub-portions of memory (Li-Cj).

18. The debugging framework of claim 17, wherein a sub-portion of memory (Li-Cj) of step a. comprises a determined number (p) of data value to store at a given time.

19. The debugging framework of claim 18, wherein the determined number (p) of data value of step a. is chosen by the user so as to enable a record of the more important data in a log item in the less intrusive way..

20. The debugging framework of claim 19, wherein the determined number (p) of data value of step a. is smaller than 10.

21. A software product, comprising the function used in the framework as claimed in any of claims 1 through 10.

« (14 pages) CABINET NETTER

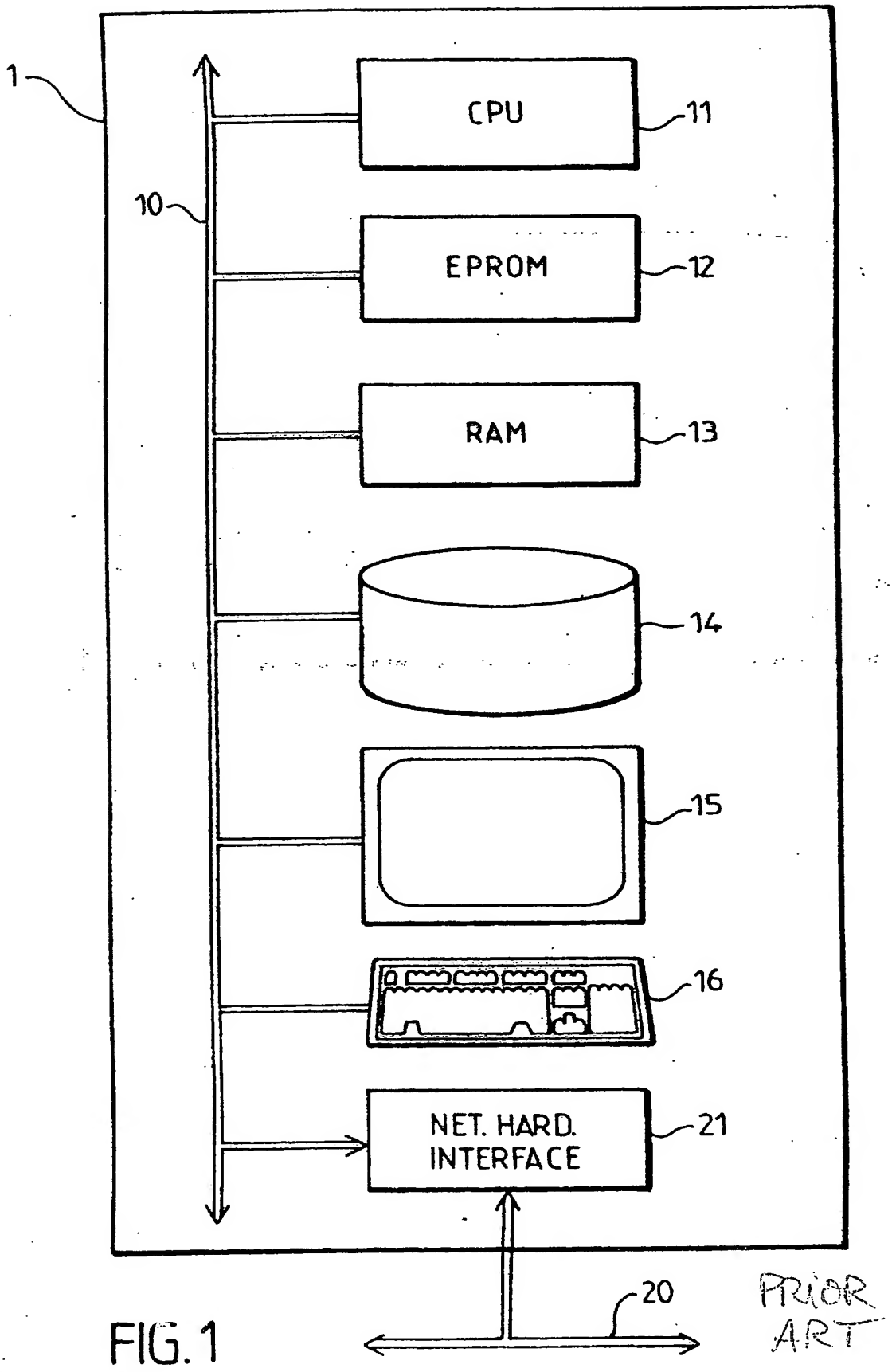


FIG. 1

PRIOR ART

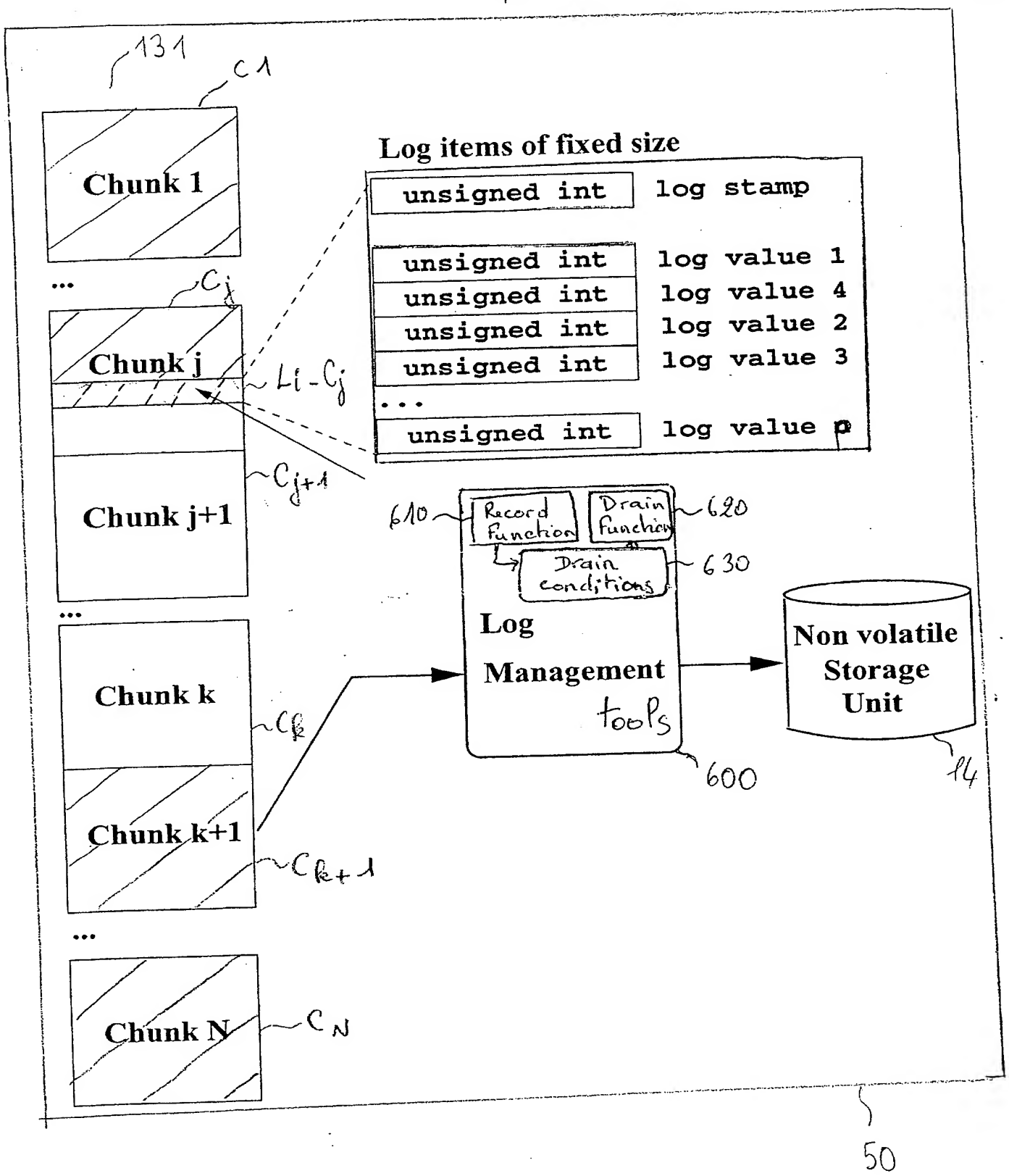


FIG 2

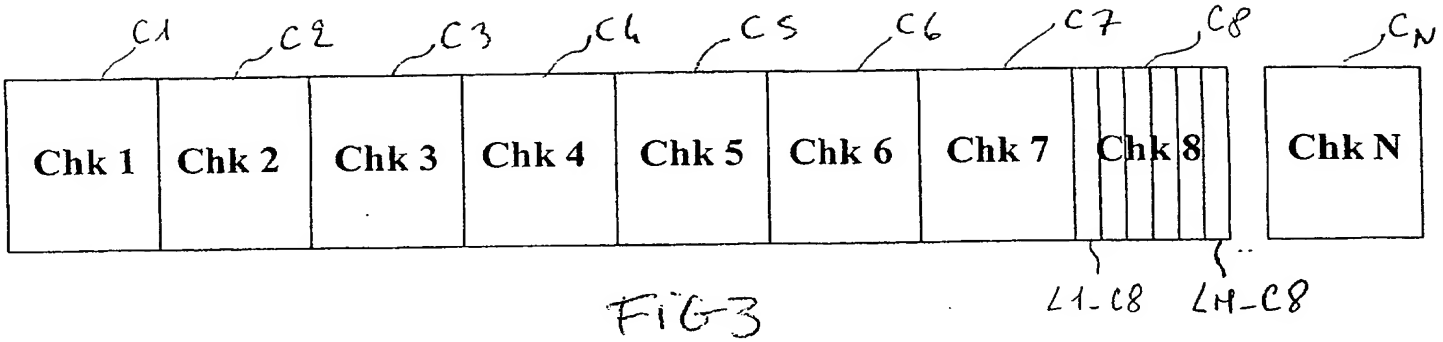


FIG3

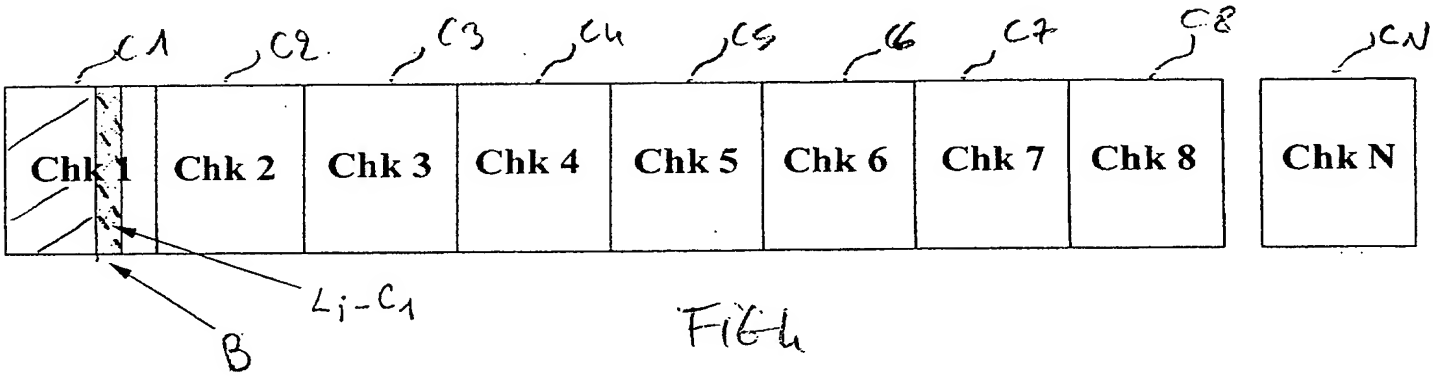


FIG4

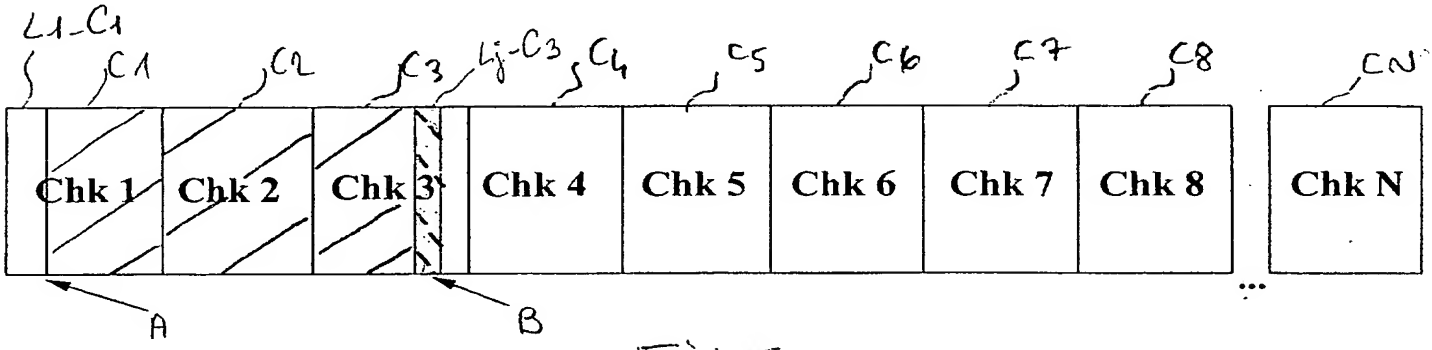


FIG5

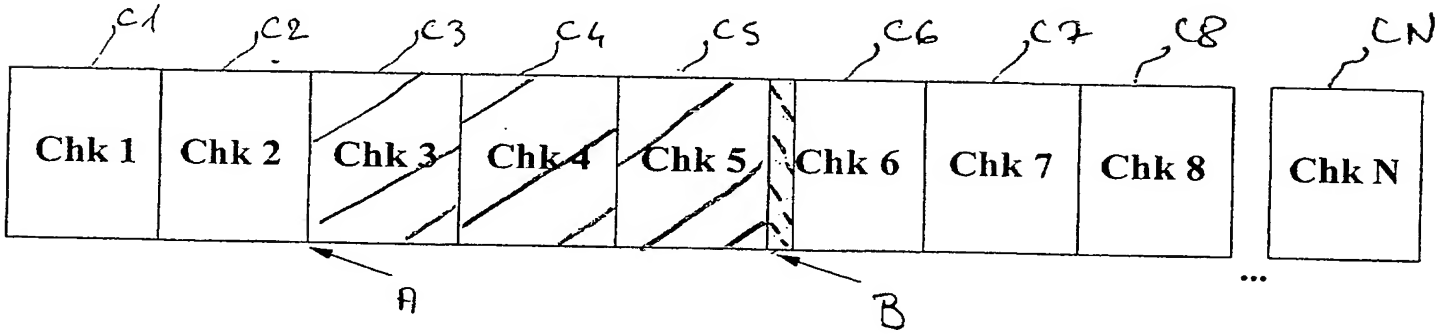


FIG6

CABINET NETTER

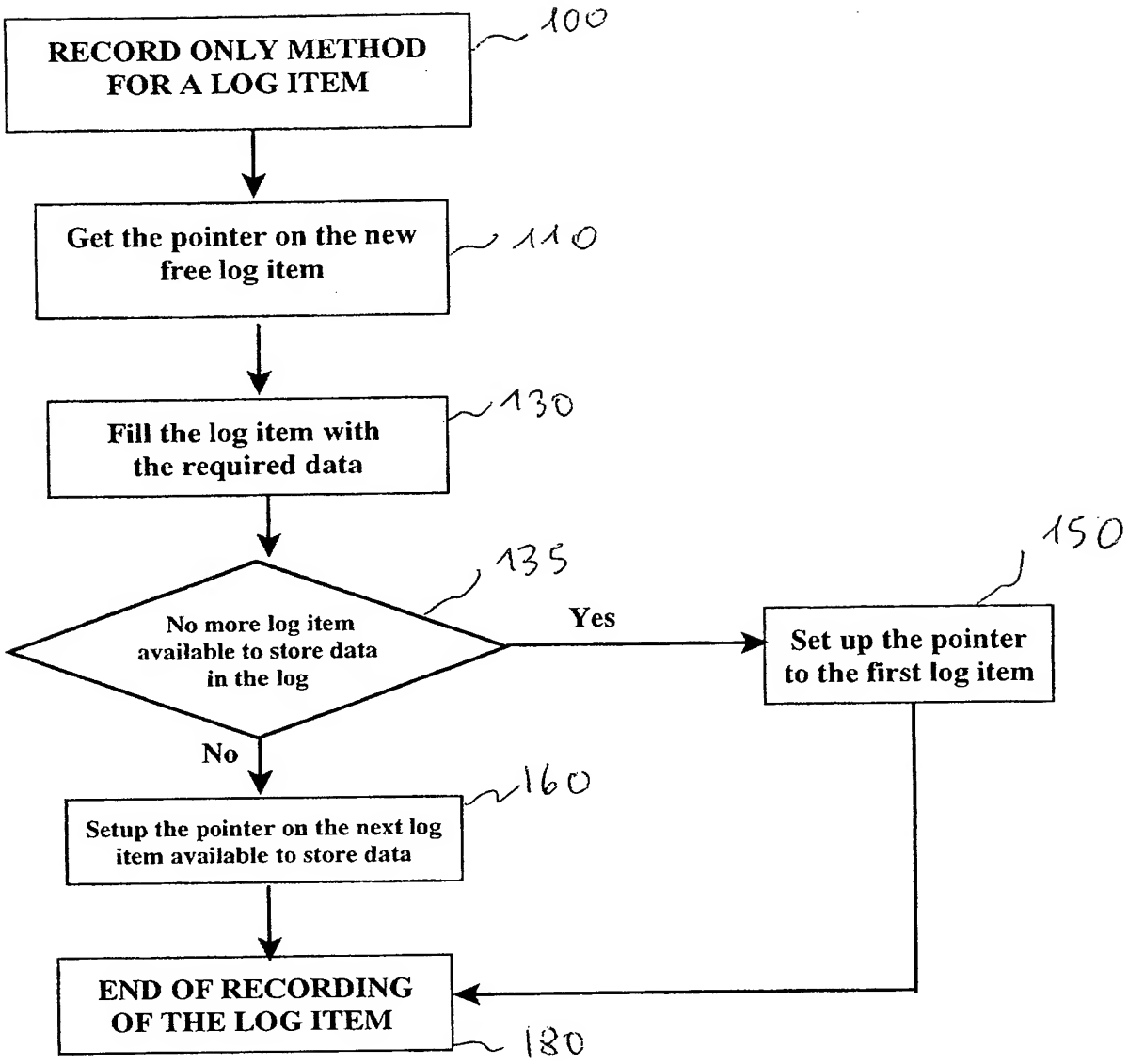
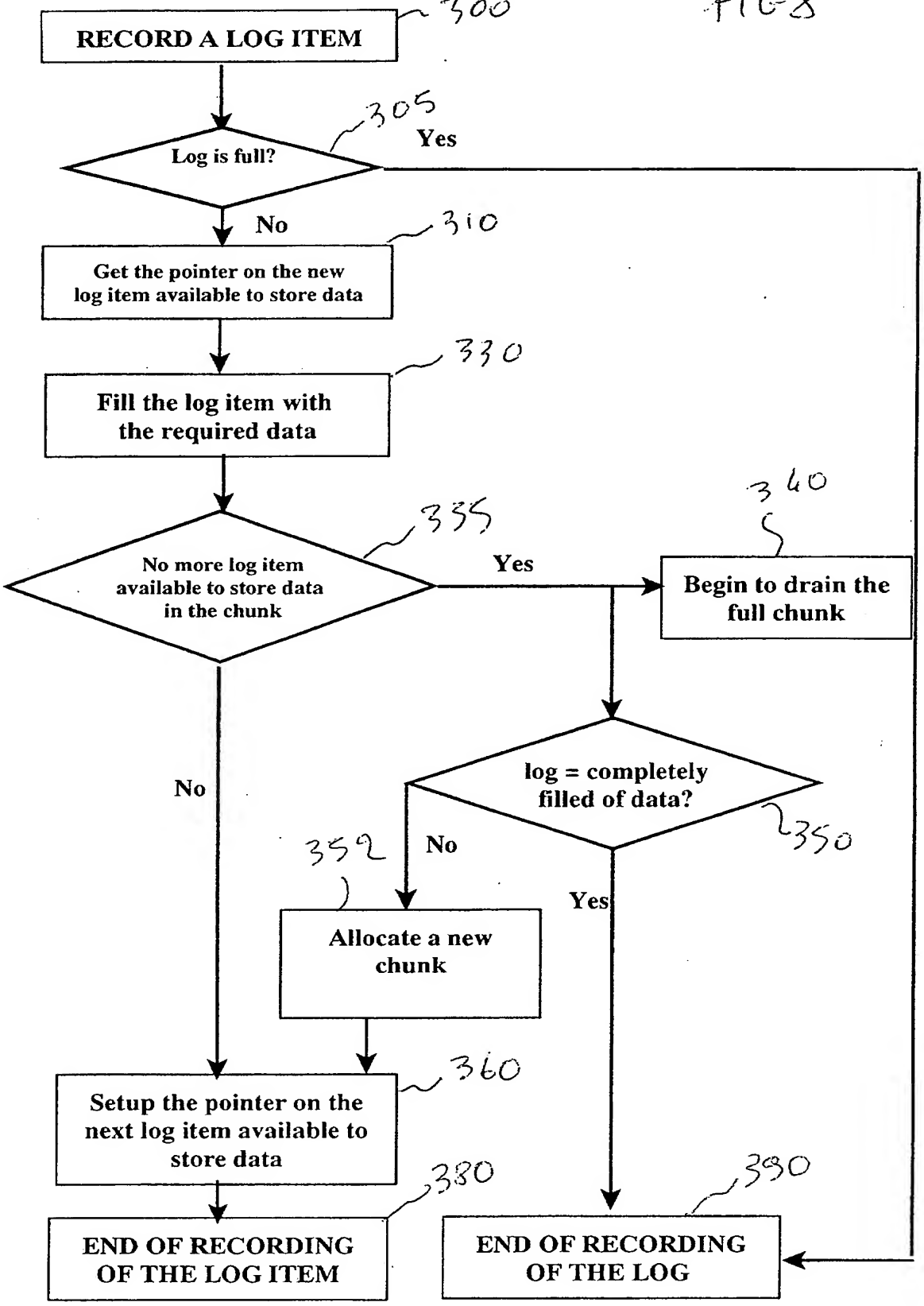


FIG 7

FIG-8



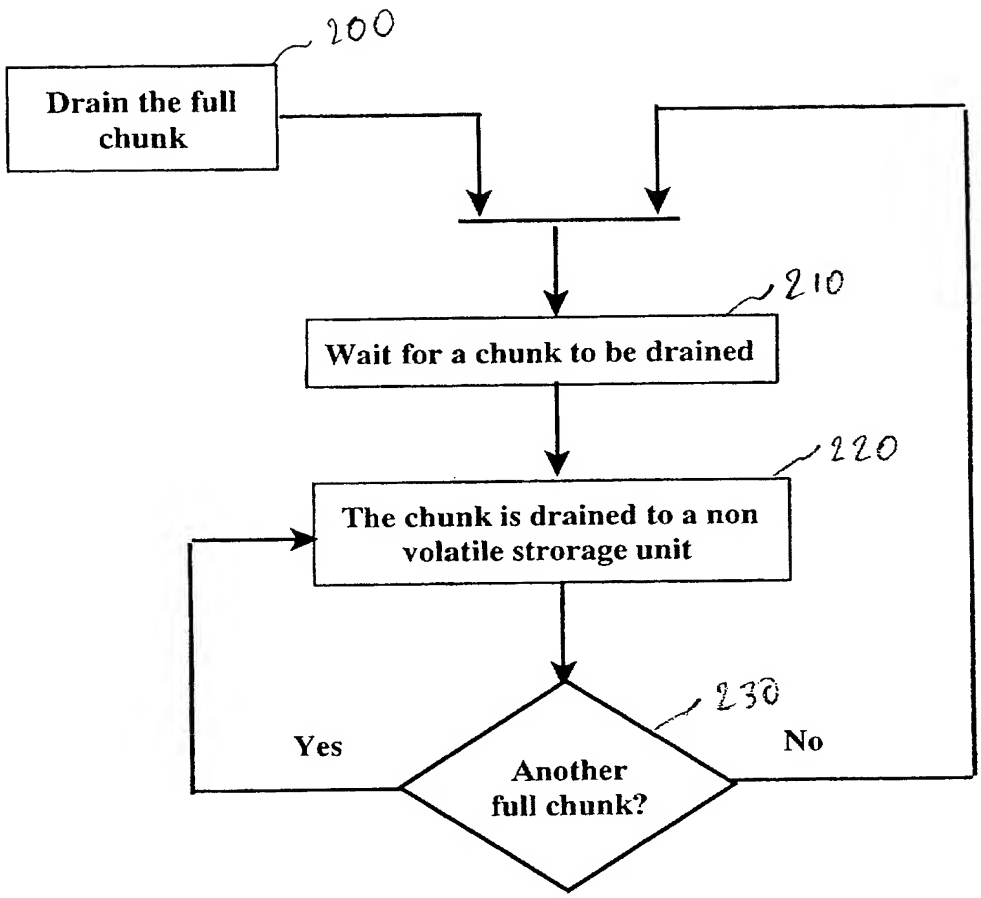


FIG 9